OpenMP
Parallel Directives for Fortran

OpenMP and Directives

OpenMP is a parallel programming system based on Fortran directives

Directives are special comments that are inserted into the source code to control parallel execution on a shared-memory machine.

All directives begin with the !OMP, COMP or *$OMP sentinel. To simplify things and avoid problems associated with the Fortran free and fixed source forms, use the !$OMP sentinel starting in column 1.

Example directives include:
!$OMP parallel
!$OMP do parallel
!$OMP end parallel

A Simple Example - Parallel Loop

```
!$OMP PARALLEL DO
        do i=1,128
                b(i) = a(i) + c(i)
        enddo
!$OMP END PARALLEL DO
```

The first directive specifies that the loop immediately following should be executed in parallel. The second directive specifies the end of the parallel section.

For codes that spend the majority of their time executing the content of simple loops, the PARALLEL DO directive can result in significant parallel performance.

Distribution of work - SCHEDULE Clause

The division of work among CPUs can be controlled with the SCHEDULE clause. For example

!$OMP PARALLEL DO SCHEDULE(STATIC)
Iterations divided among the CPUs in contiguous chunks

!$OMP PARALLEL DO SCHEDULE(STATIC, N)
Iterations divided round-robin fashion in chunks of size N

!$OMP PARALLEL DO SCHEDULE(DYNAMIC,N)
Iterations handed out in chunks of size N as CPUs become available

Example - SCHEDULE(STATIC)

CPU0:
```
do i=1,32
  a(i)=b(i)+c(i)
enddo
```

CPU1:
```
do i=3,64
  a(i)=b(i)+ c(i)
enddo
```

CPU2:
```
do i=65,96
  a(i)=b(i)+c(i)
enddo
```

CPU3:
```
do i=97,128
  a(i)=b(i)+c(i)
enddo
```

Example - SCHEDULE(STATIC,16)

```
CPU0:do i=1,16
       a(i)=b(i)+c(i)
     enddo
     do i=65,80
       a(i)=b(i)+c(i)
     enddo
CPU1:do i=17,32
       a(i)=b(i)+c(i)
     enddo
     do i=81,96
       a(i)=b(i)+c(i)
     enddo
```

```
CPU2:do i=33, 48
       a(i)=b(i)+c(i)
     enddo
     do i=97,112
       a(i)=b(i)+c(i)
     enddo
CPU3:do i=49,64
       a(i)=b(i)+c(i)
     enddo
     do i=113,128
       a(i)=b(i)+c(i)
     enddo
```

PRIVATE and SHARED Data

SHARED - variable is shared by all processors
PRIVATE - each processor has a private copy of a variable

In the previous example of a simple parallel loop, we relied on the OpenMP defaults. Explicitly, the loop would be written as

```
!$OMP PARALLEL DO SHARED(A,B,C,N) PRIVATE(I)
do I=1,N
B(I) = A(I) + C(I)
enddo
!$OMP END PARALLEL DO
```

All CPUs have access to the same storage area for A, B, C and N, but each loop needs its own private value of the loop index I.

PRIVATE Data Example

In the following loop, each processor needs its own private copy of the variable TEMP. If TEMP were shared, the result would be unpredictable since multiple processors would be writing to the same memory location.

```
!$OMP PARALLEL DO SHARED(A,B,C,N) PRIVATE(I,TEMP)
do I=1,N
TEMP = A(I)/B(I)
C(I) = TEMP + 1.0/TEMP
enddo
!$OMP END PARALLEL DO
```

REDUCTION variables

Variables that are used in collective operations over the elements of an array can be labeled as REDUCTION variables.

```
ASUM = 0.0
APROD = 1.0
!$OMP PARALLEL DO REDUCTION(+:ASUM)
REDUCTION(*:APROD)
do I=1,n
        ASUM = ASUM + A(I)
        APROD = APROD * A(I)
enddo
!$OMP END PARALLEL DO
```

Each processor has its own copy of ASUM and APROD. After the parallel work is finished, the master processor collects the values generated by each processor and performs global reduction.

The !$OMP PARALLEL directive can be used to mark entire regions as parallel. The following two examples are equivalent

```
!$OMP PARALLEL DO
do i=1,n
a(i)=b(i)+c(i)
enddo
!$OMP END PARALLEL DO
!$OMP PARALLEL DO
do i=1,n
x(i)=y(i)+z(i)
enddo
!$OMP END PARALLEL DO
```

```
!$OMP PARALLEL
!$OMP DO
do i=1,n
a(i)=b(i)+c(i)
enddo
!$OMP DO
do i=1,n
x(i)=y(i)+z(i)
enddo
!$OMP END PARALLEL
```

A more practical example using !$OMP PARALLEL

When a parallel region is exited, a barrier is implied - all threads must reach the barrier before any can proceed. By using the **NOWAIT** clause at the end of each loop inside the parallel region, an unnecessary synchronization of threads can be avoided.

```
!$OMP PARALLEL
!$OMP DO
do i=1,n
  a(i)=b(i)+c(i)
enddo
!$OMP END DO NOWAIT
!$OMP DO
do i=1,n
  x(i)=y(i)+z(i)
enddo
!$OMP END DO NOWAIT
!$OMP END PARALLEL
```

A note on OpenMP parallel loop directives

OpenMP provides two sets of directives for specifying a parallel loop. Their appropriate use is detailed below

!$OMP DO / !$OMP END DO
Used inside parallel regions marked with the PARALLEL/END PARALLEL directives

!$OMP PARALLEL DO / !$OMP END PARALLEL DO
Used to mark isolated loops outside of parallel regions

Critical Regions

Certain parallel programs may require that each processor execute a section of code, where it is critical that only one processor execute the code section at a time. These regions can be marked with the CRITICAL / END CRITICAL directives.

```
!$OMP PARALLEL SHARED(X,Y)
…
!$OMP CRITICAL (SECTION1)
call subroutine update(x)
!$OMP END CRITICAL (SECTION1)
!$OMP CRITICAL (SECTION2)
call subroutine update(y)
!$OMP END CRITICAL (SECTION2)
…
!$OMP END PARALLEL
```

More about OpenMP

Other important features of Open MP

- BARRIER - all threads must reach barrier before proceeding

- ORDERED - order of execution matches serial code

- The OpenMP runtime library - returns information regarding threads

For more complete details see the OpenMP web site at http://www.openmp.org

OpenMP Runtimes

2d optics program kernel (20 * 1024x1024 ffts with convolution)
Run on 4 processors of Cray T90 with compiler version 3.1.0.0
Run with and without OpenMP directives

source	options	CPU	Wallclock
no_omp_fft.f	none	126.9	130.3
no_omp_fft.f	-O3	110.1	111.8
no_omp_fft.f	-task3	110.2	110.4
omp_fft.f	none	123.6	38.5
omp_fft.f	-O3	111.5	34.4

OpenMP Partners

Absoft Corporation
Digital Equipment Corporation
Edinburgh Portable Compilers
GENIAS Software GmbH
Hewlett-Packard Company
Intel Corporation
International Business Machines (IBM)
Kuck & Associates, Inc. (KAI)
Myrias Computer Technologies, Inc.
Silicon Graphics, Inc. (including Cray Research)
Sun Microsystems, Inc.
The Portland Group, Inc. (PGI)